02207/12006
DC01 387103v1

TRACE CACHE BYPASSING

BACKGROUND OF THE INVENTION

Technical Field

[0001] The present invention generally relates to the management of processor instructions. More particularly, the invention relates to the selective bypassing of a trace cache build engine for enhanced performance.

Discussion

[0002] In the highly competitive computer industry, the trend toward faster processing speeds and increased functionality is well documented. While this trend is desirable to the consumer, it presents significant challenges to processor designers as well as manufacturers. A particular area of concern relates to the management of processor instructions. In modern day processor architectures, a back end allocation module executes decoded operations, typically termed micro-operations (μops), in order to implement the various features and functions called for in the program code. The front end of the processor architecture provides the μops to the allocation module, in what is often referred to as an instruction or operation pipeline. Generally, it is desirable to ensure that the front end pipeline remains as full as possible in order to optimize the processing time of the back end allocation module. As the processing speed of the allocation module increases, however, optimization becomes more difficult. As a result, a number of instruction management techniques have evolved in recent years.

[0003] FIG. 1 illustrates one such approach to managing processor instructions that involves the use of a trace cache 20. Encoded instructions 32 are provided to a decoder 22, which decodes the instructions 32 into basic μops 34 that the execution core in the back end allocation module 24 is able to execute. Since the decoding process has been found to often be a bottleneck in the process of executing instructions, one conventional approach has been to effectively recycle the retired μops 34' so that decoding is not always necessary. Thus, the retired μops 34' are sent to a build engine 26 in order to create trace data 36. The building of trace data 36 essentially involves the use of branch prediction logic and knowledge of past program execution to speculate where the program is going to execute next. Trace-based instruction caching is described in a number of sources such as U.S. Patent No. 6,170,038 to Krick, et al. The trace data 36 is written into the trace cache 20. The trace cache 20 is preferred over the decoder 22 as a source of instructions due to the above-described bottleneck concerns. For example, the time required to read from the decoder 22 is often on the order of four times longer than the time required to read from the trace cache 20. Thus, the back end allocation module 24 typically searches for a given μop in the trace cache 20 first, and resorts to the decoder 22 when the μop is not found in the trace cache 20 (i.e., a trace cache miss occurs). The difficulty with the above-described "build-at-retirement" approach is that loops in the program code may not be detected by the build engine 26 until after they are useful.

[0004] FIG. 2 illustrates another conventional approach that addresses the concerns of building at retirement, but also leaves considerable room for improvement. Under this approach, the decoded μops 34 are sent directly to a build engine 28 that includes a controller 29 that decides whether to send the trace data directly to the allocation module 24 or to the trace cache 20. Thus, when the controller 29 determines that a trace cache miss has occurred, the trace data 36' can be sent directly to the allocation module 24 in order to reduce latency. The allocation module 24 can therefore be viewed as being switched from a trace cache reading state into a build engine reading state. As trace data 36' is sent to the allocation module 24, the controller 29 can use address line 30 to determine whether it is safe to return to the trace cache read state. Specifically, as μops 34 come into the build engine 28, the controller 29 can search the trace cache 20 for the linear instruction pointer (IP) corresponding to each μop. When a match is made, the controller 29 can re-authorize the transfer of trace data 36 from the trace cache 20 to the allocation module 24. While this approach significantly helps with regard to the detection of program loops, certain difficulties remain. For example, the latency associated with the build engine 28 is part of the μop pipeline regardless of whether the trace cache 20 is being written to. Indeed, the build engine latency can become critical as build heuristics become more advanced.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The various advantages of the present invention will become apparent to one skilled in the art by reading the following specification and appended claims, and by referencing the following drawings, in which:

[0006] FIG. 1 is a block diagram of a first example of a conventional processor instruction management system, useful in understanding the invention;

[0007] FIG. 2 is a block diagram of a second example of a conventional processor instruction management system, useful in understanding the invention;

[0008] FIG. 3 is a block diagram of an example of a processor instruction management system in accordance with one embodiment of the present invention;

[0009] FIG. 4 is a block diagram of an example of a build engine controller in accordance with one embodiment of the present invention;

[0010] FIG. 5 is a block diagram of an example of a build engine controller in accordance with an alternative embodiment of the present invention;

[0011] FIG. 6 is a block diagram of an example of a back end allocation module in accordance with one embodiment of the present invention;

[0012] FIG. 7 is a flowchart of an example of a method of managing processor instructions in accordance with one embodiment of the present invention;

[0013] FIG. 8 is a flowchart of an example of a process of determining whether a resume condition is present in accordance with one embodiment of the present invention;

[0014] FIG. 9 is a flowchart of an example of a process of determining whether a resume condition is present in accordance with a first alternative embodiment of the present invention; and

[0015] FIG. 10 is a flowchart of an example of a process of determining whether a resume condition is present in accordance with a second alternative embodiment of the present invention.

DETAILED DESCRIPTION

[0016] Turning now to FIG. 3, a processor instruction management system 38 is shown. The management system 38 may be implemented in any combination of software and/or hardware known in the art. For example, one approach is to dispose each of the illustrated components on the same processor integrated circuit (chip) in the form of an application specific integrated circuit (ASIC). In other approaches, various ones of the components may reside on separate chips. While the management system 38 will be primarily described with regard to a processor used in a personal computing environment, it should be noted that the invention is not so limited. In fact, the management system 38 can be useful in any circumstance in which instruction throughput efficiency is an issue of concern. Notwithstanding, there are a number of aspects of personal computer (PC) processors for which the management system 38 is uniquely suited.

[0017] Generally, the management system 38 includes a decoder 22 to decode a first instruction into a plurality of operations. A controller 42 passes a first copy 44 of the operations from the decoder 22 to a build engine 46 associated with a trace cache 20. It can further be seen that the controller 42 also passes a second copy 48 of the operations from the decoder 22 directly to an allocation module 24 such that the operations bypass the build engine 46 and the allocation module 24 is in a decoder reading state.

[0018] It will be appreciated that the controller 42 can include control logic to determine whether a resume condition is present based on a second instruction. The second instruction can be any instruction received subsequent to receiving the first instruction, which caused the allocation module 24 to be placed in the decoder reading state. It should also be noted that the first instruction can be located anywhere in the instruction stream and is given the designation "first" only to distinguish it from the instructions that follow. The second instruction is therefore used as a mechanism of determining whether to return to the trace cache reading state (i.e., whether the resume condition is present). Simply put, the control logic is able to switch the allocation module 24 from the decoder reading state to a trace cache reading state when the resume condition is present. One approach is to search the trace cache 20 via look-up port 52 for an instruction pointer (IP) that corresponds to each following instruction. It should be pointed out that in most cache memories, a data array includes a plurality of data lines, and a tag array includes a plurality of tag entries corresponding to the data lines. Together, the tag entries and the corresponding data lines form cache lines of the trace cache 20. The above approach involves searching the tag array for each IP encountered at the decoder. While such an approach results in a high level of confidence that the trace cache 20 is ready to start providing trace data 37, searching for every instruction requires a relatively large increase to the tag structure used to implement the search of trace cache 20.

[0019] Thus, FIGs. 4 and 5 illustrate an approach wherein the control logic conducts the searching for a subset of every instruction decoded by the decoder. Specifically, FIG. 4 shows that the controller 42' may further include an abbreviated tag array 54, where the control logic 50 selects the subset based on a look-up to the abbreviated tag array 54. The abbreviated tag array 54 is constructed using build data from the build engine 46 (FIG. 3), and can contain data for the instructions that are most likely to result in a trace cache hit. FIG. 5, on the other hand, illustrates that the controller 42" may alternatively include instruction heuristics 56, where the control logic 58 selects the subset based on the instruction heuristics 56. An example of a heuristic would be to determine whether the immediately preceding instruction was a branch instruction. If not, it has been determined that the following instruction often results in a trace cache hit. Thus, in either case, if the control logic determines that the second instruction is included in the subset of instructions, the IP search is conducted in the tag array of the trace cache 20 (FIG. 3). Otherwise, the allocation module 24 remains in the decoder reading state and the decoder moves to the next instruction.

[0020] It is important to note that the decision of whether to return to the trace cache reading state should be made early enough to hide the trace cache read latency. For example, the trace cache read latency might be five pipeline stages, which means that in many conventional decoders the decision must be made by the end of the first decode stage.

[0021] FIG. 6 illustrates that one implementation of the back end allocation module 24 includes a μop que 78 and allocation engine 80. The μop que 78 provides a buffer between the front end discussed above and the allocation engine 80, if so desired.

[0022] Turning now to FIG. 7, a method 60 of managing processor instructions is shown. Generally, it can be seen that a first instruction is decoded into a plurality of operations with a decoder at processing block 62. Processing block 64 provides for passing a first copy of the operations from the decoder to a build engine associated with a trace cache. A second copy of the operations is passed from the decoder directly to a back end allocation module at block 66 such that the operations bypass the build engine and the allocation module is in a decoder reading state. A second instruction is received at block 68 and block 70 provides for determining at the decoder whether a resume condition is present based on the second instruction. It can further been seen that the allocation module is switched from the decoder reading state to a trace cache reading state at block 72 when the resume condition is present. It is important to note that when a trace cache miss is detected at block 74, the allocation module is placed in a decoder reading state at block 76 for at least one instruction.

[0023] FIG. 8 shows one approach to determining whether a resume condition is present in greater detail at block 70. It can be seen that block 82 provides for determining a linear instruction pointer (IP) for the second instruction. As already discussed, each IP provides a mechanism for addressing traces and trace segments, and the linear IP has been well documented in a number of sources. The trace cache is searched at block 84 for the IP that corresponds to the second instruction. The illustrated embodiment determines whether the IP is in the trace cache at block 86 for every instruction decoded by the decoder. If the IP is not found in the trace cache, the allocation module remains in the decoder reading state at block 88.

[0024] Turning now to FIG. 9, an alternative approach to determining whether the resume condition is present is shown in greater detail at 70'. Specifically, an IP is determined at block 82 and block 90 provides for looking up the IP in an abbreviated tag array. Thus, method 70' can be implemented via the controller 42' discussed above (FIG. 4). Processing block 92 provides for determining whether the IP is present in the abbreviated array. It will be appreciated that the abbreviated tag array can be constructed from build data to identify a subset of all instructions, where the subset includes those instructions most likely to result in a trace cache hit. If the IP is found in the abbreviated tag array, a search is performed at block 84 in the trace cache for the IP. Thus, a subset of every instruction decoded by the decoder is searched for in the trace cache, resulting in potentially significant time savings.

[0025] FIG. 10 shows another approach to determining whether the resume condition is present at block 70" in greater detail. Generally, it can be seen that the subset of instructions is selected based on an instruction heuristic instead of a look-up to an abbreviated tag array. In the illustrated embodiment, block 94 provides for determining whether an immediately proceeding instruction was a branch instruction. If not, a trace cache search is conducted for the LIP. Otherwise, block 88 provides for remaining in the decoder reading state.

[0026] Those skilled in the art can now appreciate from the foregoing description that the broad teachings of the present invention can be implemented in a variety of forms. Therefore, while this invention has been described in connection with particular examples thereof, the true scope of the invention should not be so limited since other modifications will become apparent to the skilled practitioner upon a study of the drawings, specification, and following claims.